```
import bql
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

# Connect to Bloomberg BQL service
bq = bql.Service()

# ----------------------------
# Step 1: FETCH INTRADAY DATA
# ----------------------------
tickers = {
    "TY1": "TY1 Comdty",         # 10Y Treasury Future
    "SOFR": "SR3 Comdty",        # SOFR Future
    "SWAP5Y": "USSW5 Curncy"     # 5Y USD Swap Rate
}

start_dt = "2024-06-01T09:30:00"
end_dt = "2024-06-01T16:00:00"
interval = "5MIN"

# BQL intraday queries
qry_ty = bq.execute(bql.Request(tickers["TY1"],
    bq.f.intraday("PX_LAST", interval=interval, start=start_dt, end=end_dt)))
qry_sofr = bq.execute(bql.Request(tickers["SOFR"],
    bq.f.intraday("PX_LAST", interval=interval, start=start_dt, end=end_dt)))
qry_swap = bq.execute(bql.Request(tickers["SWAP5Y"],
    bq.f.intraday("LAST_PRICE", interval=interval, start=start_dt, end=end_dt)))

# Convert to DataFrames
df_ty = qry_ty[0].df().rename(columns={"value": "TY1_price"}).set_index("time")
df_sofr = qry_sofr[0].df().rename(columns={"value": "SOFR_fut"}).set_index("time")
df_swap = qry_swap[0].df().rename(columns={"value": "SWAP5Y_rate"}).set_index("time")

# Merge and clean
df = df_ty.join(df_swap, how='outer').join(df_sofr, how='outer')
df = df.fillna(method='ffill').dropna()

# ----------------------------
# Step 2: SIMULATE EXPOSURE & DV01
# ----------------------------
np.random.seed(42)
```

```
df["notional_exposure"] = np.random.choice([10e6, 20e6, 30e6], size=len(df))
df["TY1_dv01"] = df["notional_exposure"] * (-0.0000085)
df["SWAP5Y_dv01"] = df["notional_exposure"] * 0.00005
df["SOFR_dv01"] = df["notional_exposure"] * 0.00002
df["net_dv01"] = df["TY1_dv01"] + df["SWAP5Y_dv01"] + df["SOFR_dv01"]
DV01_THRESHOLD = 500
df["hedge_needed"] = (abs(df["net_dv01"]) > DV01_THRESHOLD).astype(int)

# ----------------------------
# Step 3: TRAIN MODEL
# ----------------------------
features = ["TY1_price", "SWAP5Y_rate", "SOFR_fut", "notional_exposure"]
X = df[features]
y = df["hedge_needed"]

X_train, X_test, y_train, y_test = train_test_split(X, y, shuffle=False, test_size=0.3)

model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

df.loc[X_test.index, "hedge_pred"] = model.predict(X_test)

# ----------------------------
# Step 4: STRATEGY RECOMMENDATION
# ----------------------------
df["hedge_strategy"] = "No hedge needed"
df.loc[(df["hedge_pred"] == 1) & (df["net_dv01"] > 0), "hedge_strategy"] = "Sell TY1 or Pay 5Y
Swap"
df.loc[(df["hedge_pred"] == 1) & (df["net_dv01"] < 0), "hedge_strategy"] = "Buy TY1 or Receive
5Y Swap"

# ----------------------------
# Step 5: OUTPUT
# ----------------------------
report = classification_report(y_test, df.loc[X_test.index, "hedge_pred"], output_dict=True)
output_preview = df[["TY1_price", "SWAP5Y_rate", "SOFR_fut", "net_dv01", "hedge_pred",
"hedge_strategy"]].tail(10)

(report, output_preview)
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
import datetime

# Simulated historical intraday data (Normally, fetched via Bloomberg API)
# For demo, create synthetic dataset for 10Y Treasury Futures (TY1), 5Y Swaps, and SOFR Futures
np.random.seed(42)
minutes = pd.date_range(start="2024-06-01 09:30", end="2024-06-01 16:00", freq="5min")
n = len(minutes)

# Simulate price and DV01 sensitivity for 3 instruments
data = pd.DataFrame(index=minutes)
data["TY1_price"] = 111 + np.cumsum(np.random.normal(0, 0.03, n))
data["SWAP5Y_rate"] = 3.5 + np.cumsum(np.random.normal(0, 0.002, n))
data["SOFR_fut"] = 95 + np.cumsum(np.random.normal(0, 0.01, n))
data["notional_exposure"] = np.random.choice([10e6, 20e6, 30e6], n)  # example book size

# Calculate approximate DV01 (simplified assumptions)
data["TY1_dv01"] = data["notional_exposure"] * (-0.0000085)  # DV01 per USD notional
data["SWAP5Y_dv01"] = data["notional_exposure"] * 0.00005
data["SOFR_dv01"] = data["notional_exposure"] * 0.00002

# Net portfolio DV01 exposure
data["net_dv01"] = data["TY1_dv01"] + data["SWAP5Y_dv01"] + data["SOFR_dv01"]

# Target variable: Whether to hedge (1 = hedge needed, 0 = no hedge)
DV01_THRESHOLD = 500  # in USD
data["hedge_needed"] = (abs(data["net_dv01"]) > DV01_THRESHOLD).astype(int)

# Features for model
features = ["TY1_price", "SWAP5Y_rate", "SOFR_fut", "notional_exposure"]
X = data[features]
y = data["hedge_needed"]

# Train-test split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, shuffle=False)

# Model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# Evaluation
report = classification_report(y_test, y_pred, output_dict=True)
conf_matrix = confusion_matrix(y_test, y_pred)

# Simulate strategy recommendation
X_test["hedge_signal"] = y_pred
X_test["recommended_strategy"] = np.where(
    X_test["hedge_signal"] == 1,
    np.where(data.loc[X_test.index]["net_dv01"] > 0, "Sell TY1 or Pay 5Y Swap", "Buy TY1 or
Receive 5Y Swap"),
    "No hedge needed"
)

# Output a few rows of strategy
strategy_preview = X_test[["hedge_signal", "recommended_strategy"]].head(10)

(report, conf_matrix, strategy_preview)
```